Learning Balls of Strings with Correction Queries

Colin de La Higuera, Leonor Becerra Bonache, Jean-Christophe Janodet, Frédéric Tantini

Learning Balls of Strings with Correction Queries*
(Long Version)

Leonor Becerra Bonache[1], Colin de la Higuera[2], Jean-Christophe Janodet[2], and Frédéric Tantini[2]

[1] Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain
leonor.becerra@urv.cat
[2] Laboratoire Hubert Curien, Université Jean Monnet
18 rue du Professeur Benoît Lauras, 42000 Saint-Étienne, France
{cdlh,janodet,frederic.tantini}@univ-st-etienne.fr

Abstract. During the 80's, Dana Angluin developed an active learning paradigm, based on the use of an Oracle, able to answer different sort of queries, of which the best known are the membership and equivalence queries. However, practical evidence tends to show that if the former are often available, this is usually not the case of the latter. To get round this problem, we propose to use a new kind of queries, called *correction queries*, which we study in the framework of grammatical inference. When a string is submitted to the Oracle, either she validates it if it belongs to the target language, or she proposes a correction, that is to say, a string of the language close to the query *w.r.t.* the edit distance. Then we introduce a non-standard class of languages, that of topological balls of strings. We show that this class is not learnable in Angluin's MAT model, but that it is learnable with a linear number of correction queries. Finally, we conduct several experiments with an Oracle simulating the behaviour of a human Expert, and show that our algorithm is resistant to approximate answers.

Keywords: Grammatical Inference, Oracle Learning, Correction Queries, Edit Distance, Balls of Strings.

1 Introduction

Do you know how many *Nabcodonosaur* were kings of Babylon? And do you know when *Arnold Shwartzeneger* was born? A few years ago, just 2 decades ago, you would have had to consult encyclopedias and Who's Who dictionaries in order to get answers to such questions. At that time, you may have needed

* This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

this information in order to participate to quizzes and competitions organised by famous magazines during the summers, but because of *these* questions, you might possibly have missed the very first prize. Why?... Nowadays, everything has changed: You naturally use the Web, launch your favourite search engine, type 2 keywords, follow 3 links and note down the answers. In this particular case, you discover... that no king of Babylon was called *Nabcodonosaur* but 2 *Nab***u**ch*odonos***o***r*'s reigned there many centuries ago. Again, the day *Arnold Shwartzeneger* was born is not clear, but it is easy to check that *Arnold S***c***hwa***rz***ene***gg***er* was born in 1947, July 30^{th}.

So you would probably win today the great competitions of the past. Indeed, the actual search engines are able to propose *corrections* when a keyword is not frequent. Those corrections are most often reliable because the reference dictionary is built from the billions of web pages indexed all over the world. Hence, the Web and the search engines are playing the part of an imperfect but powerful Oracle, able to return relevant documents from a relevant query, but also to correct any suspect query. In other words, such an oracle is able to answer to what we shall call *correction queries*.

The setting itself occurs in other circumstances: When a human being is asked to provide data in an interactive situation, an alternative to having the human expert labelling huge quantities of data can be to have the learning system ask (query) the human expert who then only labels those items that are required. Nevertheless, assuming that the Oracle is a human expert introduces new constraints. On the one hand, it is inconceivable to ask *a polynomial* number of queries: There may be no chance to get enough answers in reasonable time. So the learning algorithm should aim at minimising the number of queries even if we must pay for it with a worse time complexity. On the other hand, a human (or even the Web) is fallible. Therefore the learning algorithm should aim at learning functions or languages that are robust from corrections that may not be ideal, thus approximate.

In the above Web example, the distance used by the search engine to find a closest string is a variant of the *edit distance* which measures the minimum number of deletion, insertion or substitution operations needed to transform one string into another [1, 2]. This distance and variants where each elementary operation may have a different weight have been used in many fields including Computational Biology [3], Language Modelling [4] and Pattern Recognition [5]. In Grammatical Inference (GI), it is mainly used in 2 kinds of papers.

On the one hand, several works aim at developing standard probabilistic GI techniques in order to learn the weights of the edit operations [6, 7]. On the other hand, the edit distance naturally appears in specific GI problems, in particular when one wants to learn languages from noisy data [8, 9]. In the latter case, we must remark that the classes of languages studied there are not defined following the Chomsky Hierarchy. Indeed, even the easiest level of this hierarchy, the class of regular languages, is not at all robust to noise, since the parity functions (which can be defined as regular languages) are not learnable in the presence of noise [10]. In this paper also, in order to avoid this difficulty, we shall

consider only special finite languages, that seem elementary to formal language theoreticians, but are relevant for topologists and complex for combinatorialists: the *balls of strings*.

In the paper, we thus study the problem of identifying balls from correction queries. We begin by revisiting the edit distance and introducing the balls in Sect. 2. Then we show in Sect. 3 that balls are not learnable with Angluin's membership and equivalence queries, that is another reason to introduce the correction queries. We show in Sect. 4 that balls are learnable with a linear number of such queries. Then in Sect. 5, we study the effectiveness of our algorithm from an experimental point of view, showing that it is *robust*, in particular when the answers of the Oracle are approximate. Finally, we conclude in Sect. 6.

2 On Balls of Strings as Languages

An *alphabet* Σ is a finite nonempty set of symbols called *letters*. A *string* $w = a_1 \dots a_n$ is any finite sequence of letters. We write Σ^* for the set of all strings over Σ and λ for the empty string. Let $|w|$ be the length of w and $|w|_a$ the number of occurrences of a in w. We say that a string u is a *subsequnce* of v, denoted $u \preceq v$, if $u = a_1 \dots a_n$ and there exist $u_0, \dots, u_n \in \Sigma^*$ such that $v = u_0 a_1 u_1 \dots a_n u_n$. A *language* is any subset $L \subseteq \Sigma^*$. Let $\mathbb{N}$ be the set of non negative integers. For all $k \in \mathbb{N}$, let $\Sigma^k = \{w \in \Sigma^* : |w| = k\}$ and $\Sigma^{\leq k} = \{w \in \Sigma^* : |w| \leq k\}$.

The *edit distance* $d(w, w')$ is the minimum number of *edit operations* needed to transform w into w' [1]. More precisely, we say that w *rewrites to* w' *in 1 step*, written $w \to w'$, if either (1) $w = uav$ and $w' = uv$ (*deletion*), or (2) $w = uv$ and $w' = uav$ (*insertion*), or (3) $w = uav$ and $w' = ubv$ (*substitution*), where $u, v \in \Sigma^*$, $a, b \in \Sigma$ and $a \neq b$. Let $\xrightarrow{k}$ denote k rewriting steps. The edit distance $d(w, w')$ is the minimum $k \in \mathbb{N}$ such that $w \xrightarrow{k} w'$. *E.g.*, $d(abaa, aab) = 2$ since $a\underline{b}aa \to aa\underline{a} \to aab$ and the rewriting of $abaa$ into aab cannot be achieved with less than 2 steps. Notice that $d(w, w')$ can be computed in time $\mathcal{O}(|w| \cdot |w'|)$ by means of dynamic programming [2].

The following standard property states that $d(w, w')$ is at least the number of insertions needed to equalise the lengths: Not in ECML

Proposition 1. *For all* $w, w' \in \Sigma^*$, $d(w, w') \geq \big||w| - |w'|\big|$. *Moreover,* $d(w, w') = \big||w| - |w'|\big|$ *iff* $(w \preceq w'$ *or* $w' \preceq w)$. Not in ECML

It is well-known that the edit distance is a *metric* [11], so it conveys to Σ^* the structure of a *metric space*. Therefore, it is natural to introduce balls of strings, since in a way, these are the simplest languages one may consider. The *ball of centre* $o \in \Sigma^*$ *and radius* $r \in \mathbb{N}$, denoted $B_r(o)$, is the set of all strings whose distance is at most r from o: $B_r(o) = \{w \in \Sigma^* : d(o, w) \leq r\}$. *E.g.*, if $\Sigma = \{a, b\}$, then $B_1(ba) = \{a, b, aa, ba, bb, aba, baa, bab, bba\}$ and $B_r(\lambda) = \Sigma^{\leq r}$ for all $r \in \mathbb{N}$.

The latter example illustrates the fact that the number of strings in a ball can be exponential with the radius. Experimentally (see Table 1), we clearly notice that for a fixed length of centre, the average number of strings is more than Not in ECML

twice bigger when the radius is increased by 1. This combinatorial explosion occurs as soon as $|\Sigma| \geq 2$, although we leave open the question of finding a general formula that would give the volume of any ball $B_r(o)$. This remark

Table 1. Average number of strings in a ball with a 2 letters alphabet. Centres are random strings, each computation is done over 20 centres.

Length of the centre	Radius					
	1	2	3	4	5	6
0	3.0	7.0	15.0	31.0	63.0	127.0
1	6.0	14.0	30.0	62.0	126.0	254.0
2	8.6	25.6	56.5	119.7	246.8	501.6
3	10.8	41.4	101.8	222.8	468.6	973.0
4	13.1	61.4	173.8	402.9	870.9	1850.8
5	16.3	91.0	285.1	698.5	1584.4	3440.9
6	17.9	125.8	441.2	1177.5	2771.3	6252.9
7	21.2	166.9	678.0	1908.8	4835.8	11233.5
8	24.3	200.2	1034.2	3209.9	8358.1	19653.6
9	26.0	265.4	1390.9	5039.6	13677.8	34013.1

Not in ECML

raises the problem of the representation scheme that we should use to learn the balls. Basically, we need representations whose size is reasonable, which is not the case of an exhaustive enumeration. An alternative representation could be by *deterministic finite automata* (DFA) since balls are finite and thus regular languages. However, experiments show that the corresponding minimum DFA is often exponential with r (but linear with $|o|$) [12]. Again, a formal proof of this property is a challenging combinatorial problem, but in any case it seems that DFA are not reasonable representations of balls.

On the other hand, why not represent the ball $B_r(o)$ by the pair (o, r) itself? Indeed, its size is $|o| + \log r$. Moreover, deciding whether $w \in B_r(o)$ or not is immediate: One only has to (1) compute $d(o, w)$ and (2) check whether this distance is $\leq r$, which is achievable in time $\mathcal{O}(|o| \cdot |w| + \log r)$. Finally, it is possible for a ball to have several such representations. If $\Sigma = \{a\}$, then $B_2(a) = B_3(\lambda) = \{\lambda, a, aa, aaa\}$ for instance, but this is a special case: When the alphabet has at least 2 letters, (o, r) is a unique thus *canonical* representation of $B_r(o)$:

Theorem 1. *If* $|\Sigma| \geq 2$ *and* $B_{r_1}(o_1) = B_{r_2}(o_2)$*, then* $o_1 = o_2$ *and* $r_1 = r_2$*.*

Not in ECML

Proof. **Claim 1**: *If* $B_{r_1}(o_1) = B_{r_2}(o_2)$*, then* $|o_1| + r_1 = |o_2| + r_2$. Indeed, let $w \in \Sigma^{r_1}$, then $d(o_1, o_1 w) = |w| = r_1$ by Prop. 1, so $o_1 w \in B_{r_1}(o_1)$, thus $o_1 w \in B_{r_2}(o_2)$, *i.e.*, $d(o_1 w, o_2) \leq r_2$. Now $d(o_1 w, o_2) \geq |o_1 w| - |o_2|$ from Prop. 1. So we deduce that $r_2 \geq |o_1 w| - |o_2| = |o_1| + r_1 - |o_2|$. The same reasoning yields $|o_1| + r_1 \geq |o_2| + r_2$. **Claim 2**: *If* $|\Sigma| \geq 2$ *and* $o_2 \not\preceq o_1$*, there exists* $w \in \Sigma^*$ *such that (1)* $|w| = r_1 + |o_1|$*, (2)* $o_1 \preceq w$ *and (3)* $o_2 \not\preceq w$. Indeed, suppose that o_2 starts with an a and let $b \in \Sigma \setminus \{a\}$; We define $w = b^{r_1} o_1$ and get the result. **Theorem itself**: Assume that $o_1 \neq o_2$. Then either $o_1 \not\preceq o_2$, or $o_2 \not\preceq o_1$. Suppose the second case, without loss of generality. By Claim 2, there exists a string w such that (1) $|w| = r_1 + |o_1|$, (2) $o_1 \preceq w$ and (3) $o_2 \not\preceq w$. As $o_1 \preceq w$, we deduce that

$d(o_1, w) = |w| - |o_1|$, by Prop. 1. Moreover, as $|w| = r_1 + |o_1|$, we get $d(o_1, w) = r_1$. So $w \in B_{r_1}(o_1)$. On the other hand, $o_2 \not\preceq w$, so we get $d(o_2, w) > \big||w| - |o_2|\big|$, by Prop. 1. As $|w| = r_1 + |o_1|$, we deduce that $d(o_2, w) > \big|r_1 + |o_1| - |o_2|\big| = r_2$, so $w \notin B_{r_2}(o_2)$, thus $B_{r_1}(o_1) \neq B_{r_2}(o_2)$. This is impossible, so $o_1 = o_2$, and then $r_1 = r_2$, by Claim 1. □ Not in ECML

Hence, representing the ball $B_r(o)$ by the pair (o, r) is reasonable. However, it is worth noticing that *huge* balls, whose radii are not polynomially related to the length of the centres (*e.g.*, $r > 2^{|o|}$), will pose tricky problems of complexity. For instance, to learn the ball $B_r(\lambda) = \Sigma^{\leq r}$, one needs to manipulate at least one string of length r. Therefore, in the following, we will always consider *good* balls only:

Definition 1. *Given any fixed polynomial $q()$, we say that a ball $B_r(o)$ over an alphabet Σ is q-good if $r \leq q(|o|)$. Let $\mathcal{GB}_q(\Sigma)$ denote the set of all q-good balls.*

3 Learning Balls from Queries

Query learning is a paradigm introduced by Angluin [13]. Her model brings a Learner (he) and an Oracle (she) into play. The goal of the Learner is to identify the representation of an unknown language, by submitting queries to the Oracle. The latter knows the target language and answers properly to the queries (*i.e.*, she does not lie). Moreover, the Learner is bound by efficiency constraints: (1) He can only submit a polynomial number of queries (in the size of the target representation) and (2) the available overall time must be polynomial in the size of the target representation[3].

Between the different combinations of queries, one, called MAT (Minimally Adequate Teacher), is sufficient to learn DFA [14]. Two kinds of queries are used:

Definition 2. *Let $\Re$ be a class of languages on Σ^* and $L \in \Re$. In the case of membership queries, the Learner submits a string $w \in \Sigma^*$ to the Oracle; Her answer, denoted* MQ(w)*, is either* YES *if $w \in L$, or* NO *if $w \notin L$. In the case of equivalence queries, the Learner submits (the representation of) a language $K \in \Re$ to the Oracle; Her answer, denoted* EQ(K)*, is either* YES *if $K = L$, or a string belonging to the symmetric difference $K \triangle L$ if $K \neq L$.*

Although MQ and EQ have established themselves as a standard combination, there are real grounds to believe that EQ are too powerful to exist or even be simulated. As suggested in [14] we may be able to substitute them with a random draw of strings that are then submitted as MQ (*sampling*), but there are many cases where sampling is not possible as the relevant distribution is unknown and/or inaccessible [15]. Besides, we will not consider MQ and EQ because they do not help to learn balls:

[3] The time complexity usually concerns the time spent after receiving each new example, and takes the length of the information returned by the Oracle into account; Thus, our constraint is somewhat stronger but not restrictive, since we focus on good balls only.

Theorem 2. *Assume* $|\Sigma| \geq 2$. *Let* $m, n \in \mathbb{N}$ *and* $\mathcal{B}_{\leq m,n} = \{B_r(o) : r \leq m, o \in \Sigma^*, |o| \leq n\}$. *Any algorithm that identifies every ball of* $\mathcal{B}_{\leq m,n}$ *with* EQ *and* MQ *necessarily uses* $\Omega(|\Sigma|^n)$ *queries in the worst case.*

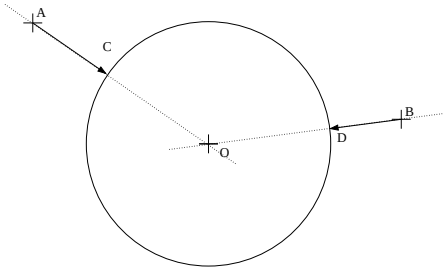
Proof. Following [16], we describe an Adversary who maintains a set S of all possible balls. At the beginning, $S = \mathcal{B}_{\leq m,n}$. Her answer to the equivalence query $L = B_r(o)$ is a counterexample o. Her answer to the membership query o is NO. At each step, the Adversary eliminates many balls of S but only one of centre o and radius 0. As there are $\mathcal{O}(|\Sigma|^n)$ such balls in $\mathcal{B}_{\leq m,n}$, identifying them requires $\Omega(|\Sigma|^n)$ queries. □

It should be noted that if the Learner is given one string from the ball, he can learn using a polynomial number of MQ. We shall see that *correction queries* (CQ), introduced below, allow to get round these problems:

Definition 3. *Let* L *be a fixed language and* w *a string submitted by the Learner to the Oracle. Her answer, denoted* $\text{CQ}(w)$*, is either* YES *if* $w \in L$*, or a correction of* w *w.r.t.* L *if* $w \notin L$*, that is a string* $w' \in L$ *at minimum edit distance from* w*:* $\text{CQ}(w) =$ *one string of* $\{w' \in L : d(w, w') \text{ is minimum}\}$.

Not in ECML : will be shorten

Notice that the CQ can easily be simulated knowing the target language. Moreover, we have seen in the introduction that they naturally exist in real-world applications such as the search engines of the Web. Also, we can note that CQ are relevant from a cognitive point of view: There is growing evidence that a child acquires his native language using the corrections of his mother [17]. And last but not least, CQ as well as balls rely on a distance, that foreshadows nice learning results. Indeed, suppose that we are not working with the edit distance and strings, but the Euclidean distance and disks in the plane. We can proceed in 3 stages to learn a disk of centre O and radius R with CQ.



(1) We start by selecting 2 points A, B far from each other and outside of the disk we want to identify. Looking for them haphazardly by asking to the Oracle if such or such a point is in the disk is enough: Intuitively, we are going to find them with very few queries. (2) We ask the Oracle to give corrections to A and B. Concerning A, the Oracle returns a point C inside the disk, as close as possible to A. Clearly, this point will be at the intersection of the segment $[OA]$ and the boundary circle of the target disk. Likewise, let D be the correction of B. (3) We draw the lines (AC) and (BD) with a ruler: They intersect in O. Then we draw the circle with a compass. We get the radius by measuring the distance between O and C. Hence, it is easy to learn the balls of $\mathbb{R}^2$ with CQ. Now, focusing on balls of strings, we may hope that the previous approach is good and try to reproduce it.

Not in ECML

4 Identifying Balls using Corrections

In this section, we propose an algorithm that learns balls using a *linear* number of CQ. Actually, we are going to follow the previous algorithm, that learns balls of $\mathbb{R}^2$ by searching border points in order to deduce the centre. Nevertheless, the problem is more complicated with balls of strings.

First, when one submits a string outside of a ball to the Oracle, she still answers with a string that belongs to the circle delimiting the ball. In other words, it will still be possible to circumscribe it. However, a string often has a lot of different possible corrections, contrarily to what happens in the plane. *E.g.*, the possible corrections for the string $aaaa$ *w.r.t.* the ball $B_2(bb)$ are $\{aa, aab, aba, baa, aabb, abab, abba, baab, baba, bbaa\}$. By definition of the CQ, the Oracle will choose one of them arbitrarily, potentially the worst one *w.r.t.* the Learner's point of view. Nevertheless, the Oracle's potential malevolence is limited by the following result, that characterises the set of *all* the possible corrections for a string:

Theorem 3. *Let $B_r(o)$ be a ball and m a string s.t. $m \notin B_r(o)$. Then the set of possible corrections of m is exactly $\{z \in \Sigma^* : d(o,z) = r$ and $d(z,m) = d(o,m) - r\}$.*

Proof. Let $k = d(o,m)$ and consider a derivation from o to m of minimum length: $o \xrightarrow{k} m$. As $m \notin B_r(o)$, we get $k > r$, so this derivation passes through a string w_0 such that $o \xrightarrow{r} w_0 \xrightarrow{k-r} m$. Let us define the set $W = \{w \in \Sigma^* : d(o,w) = r$ and $d(w,m) = k - r\}$. Basically, $w_0 \in W$, so $W \neq \emptyset$. Moreover, $W \subseteq B_r(o)$. Finally, let Z denote the set of all the possible corrections of m. We claim that $Z = W$. Indeed, let $z \in Z$ and $w \in W$. If $d(z,m) > d(w,m)$, then w is a string of $B_r(o)$ that is strictly closer to m than z, so z cannot be a correction of m. On the other hand, if $d(z,m) < d(w,m)$, then $d(o,m) \leq d(o,z) + d(z,m)$, and it follows that $d(o,z) \geq d(o,m) - d(z,m) > d(o,m) - d(w,m)$. As $d(o,m) = k$ and $d(w,m) = k - r$, we get $d(o,z) > r$, which is impossible since $z \in Z \subseteq B_r(o)$. So $d(z,m) = d(w,m) = k - r$. In consequence, every string $w \in W$ is at the same distance from m as every correction $z \in Z$, thus $W \subseteq Z$. Moreover, we have $d(o,m) \leq d(o,z) + d(z,m)$, so $k \leq d(o,z) + k - r$, so $d(o,z) \geq r$. As $z \in B_r(o)$, we deduce that $d(o,z) = r$. Moreover, we have stated that $d(z,m) = k - r$, so we conclude that $Z \subseteq W$. □ Not in ECML

Not in ECML

Here is a geometric interpretation of the result above. Let us define the *segment* $[o,m] = \{w \in \Sigma^* : d(o,w) + d(w,m) = d(o,m)\}$ and the *circle* $C_r(o) = \{w \in \Sigma^* : d(o,w) = r\}$. Theo. 3 states that a string z is a possible correction of m *iff* $z \in [o,m] \cap C_r(o)$. The fact that m has several possible corrections shows that the geometry of Σ^* is very different from that of $\mathbb{R}^2$.

Secondly, building the centre of a ball from strings on its periphery is difficult for at least 2 reasons. On the one hand, (Σ^*, d) is a metric space with no vector space as an underlying structure. This is the same story as if we were trying to learn the disks of the plane with just a compass but no ruler... On the other hand, the problem is formally *hard* [18]:

Theorem 4. *Given a finite set of strings $W = \{w_1, \ldots, w_n\}$, finding $z \in \Sigma^*$ that minimises $\sum_{w \in W} d(z, w)$ or $\max_{w \in W} d(z, w)$, are $\mathcal{NP}$-hard problems.*

Therefore, we must study the balls in more detail and make the best possible use of the CQ so as not to build the centres from scratch. We begin by distinguishing the longest strings of any ball:

Definition 4. *The upper border of a ball $B_r(o)$, denoted $B_r^{max}(o)$, is the set of all the strings that belong to $B_r(o)$ and are of maximum length: $B_r^{max}(o) = \{z \in B_r(o) : \forall w \in B_r(o), |w| \leq |z|\}$.*

E.g., let $\Sigma = \{a, b\}$. As $B_1(ba) = \{a, b, aa, ba, bb, aba, baa, bab, bba\}$, we get $B_1^{max}(ba) = \{aba, baa, bab, bba\}$. The strings of $B_r^{max}(o)$ are remarkable because they are all built from the centre o by doing r insertions. So having one string $w \in B_r^{max}(o)$, one 'simply' has to guess the inserted letters and delete them to find o again. We get:

Not in ECML

Proposition 2. *$w \in B_r^{max}(o)$ iff ($o \preceq w$ and $d(o, w) = |w| - |o| = r$).*

Proof. Let us assume that $o \preceq w$ and $d(o, w) = |w| - |o| = r$. Then $w \in B_r(o)$. Let w' be a string *s.t.* $|w'| > |w|$. Then, by Prop. 1, $d(o, w') \geq |w'| - |o| > |w| - |o| = r$, so $w' \notin B_r(o)$. Therefore, $w \in B_r^{max}(o)$. Conversely, let $w \in B_r^{max}(o)$ and $k = d(o, w) \leq r$. Then there exists a rewriting derivation $o \xrightarrow{k} w$. If a deletion appears along this derivation, then forgetting it leads to a new derivation from o to some string w' whose length is $|w| + 1$ and whose distance to o is $\leq k$, so $w \notin B_r^{max}(o)$. If a substitution appears along the derivation, say an a by a b, then keeping the a and making the insertion of a new b also leads to a derivation from o to some string w' whose length is $|w| + 1$ and distance to o is $\leq k$, so $w \notin B_r^{max}(o)$. Therefore, only insertions occur along the derivation, so $o \preceq w$. Lastly, if $k < r$, then doing one more insertion leads to a derivation from o to some w' whose length is $|w| + 1$ and distance to o is $\leq k$, so $w \notin B_r^{max}(o)$. Hence $k = r$. □

Not in ECML

Several strings of $B_r^{max}(o)$ are very informative. Indeed, let $a \in \Sigma$ be an arbitrary letter. Then the string $a^r o = \underbrace{a \ldots a}_{r \text{ times}} o$ satisfies $o \preceq a^r o$ and $d(o, a^r o) = r$, so $a^r o \in B_r^{max}(o)$. Moreover, if one knows r, one can easily deduce o. We claim that CQ allow us to get hold of $a^r o$ from any string $w \in B_r^{max}(o)$ by swapping the letters (see Algo. 1).

For instance, $B_2^{max}(bb) = \{aabb, abab, abba, abbb, baab, baba, babb, bbaa, bbab, bbba, bbbb\}$. Running EXTRACT_CENTRE on the string $w = baba$ and radius $r = 2$ transforms, at each loop, the i^{th} letter of w to an a that is put at the beginning and then submits it to the Oracle. c counts the number of times this transformation is accepted. We get:

i	w	w'	CQ(w')	w changes	c
1	$\underline{b}aba$	$aaba$	$baba$	no	0
2	$b\underline{a}ba$	$abba$	YES	yes	1
3	$ab\underline{b}a$	$aabb$	YES	yes	2

Algorithm 1 EXTRACT_CENTRE

Require: a string $w = x_1 \dots x_n \in B_r^{max}(o)$, the radius r
Ensure: the centre o of the ball $B_r(o)$
1: $c \leftarrow 0; i \leftarrow 1; a \leftarrow x_n$
2: **while** $c < r$ **do**
3: Assume $w = x_1 \dots x_n$ and let $w' = a x_1 \dots x_{i-1} x_{i+1} \dots x_n$
4: **if** CQ(w') = YES **then** $w \leftarrow w'; c \leftarrow c + 1$ **end if**
5: $i \leftarrow i + 1$
6: **end while**
7: Assume $w = x_1 \dots x_n$ and **return** $x_{r+1} \dots x_n$

When $c = 2 = r$, EXTRACT_CENTRE stops with $w = aabb$ and returns $o = bb$.

Lemma 1. *Algo. 1 is correct: Given r and $w \in B_r^{max}(o)$,* EXTRACT_CENTRE *returns o using $\mathcal{O}(|o| + r)$* CQ *and a polynomial amount of time.*

Proof. Let us show that the swapping operation is correct. Let $w = x_1 \dots x_n \in B_r^{max}(o)$ and $w' = a x_1 \dots x_{i-1} x_{i+1} \dots x_n$ for some $i \in 1..n$. If there exists at least one derivation $o \xrightarrow{r} w$ where the letter x_i of w comes from an insertion in o, then deleting x_i and doing the insertion of an a in front of o yields a string w' that satisfies $o \preceq w'$ and $|w'| = |w|$. By Prop. 1, we get $d(o, w') = |w'| - |c| = |w| - |c| = r$, so by Prop. 2, $w' \in B_r^{max}(o)$ and CQ(w') = YES. On the other hand, if there is no derivation where x_i is introduced by an insertion, then x_i is a 'real' letter of o. In this case, deleting x_i and inserting an a yields a string w' *s.t.* $o \not\preceq w'$. By Prop. 1, we get $d(o, w') > |w'| - |o|$. As $|w'| = |w|$, we deduce that $d(o, w') > r$. So $w' \notin B_r^{max}(o)$ and CQ$(w') \neq$ YES. □

Not in ECML : We should justify the correctness of the swapping operation in the text.

Not in ECML

Hence, we are now able to deduce the centre of a ball as soon as we know its radius and a string from its upper border. The following technical lemma is a step towards finding this string (although we have no information about r and $|o|$ yet):

Lemma 2. *Suppose $\Sigma = \{a_1, \dots, a_n\}$. Then every correction w of the string $m = (a_1 \dots a_n)^k$ where $k \geq |o| + r$ belongs to $B_r^{max}(o)$.*

Proof. Let Z be the set of all the possible corrections of m. Let us show that $Z = B_r^{max}(o)$. As $m = (a_1 \dots a_n)^k$ with $k \geq |o| + r$, we get $o \preceq m$, so by Prop. 1, $d(o, m) = |m| - |o|$. Let $w \in B_r^{max}(o)$. By Prop. 2, we get $o \preceq w$ and $d(o, w) = |w| - |o| = r$. Moreover, as $m = (a_1 \dots a_n)^k$ with $k \geq |o| + r$, we get $w \preceq m$, so by Prop. 1, $d(w, m) = |m| - |w| = |m| - |o| - r = d(o, m) - r$. As $d(o, w) = r$ and $d(w, m) = d(o, m) - r$, we deduce that $B_r^{max}(o) \subseteq Z$, by Theo. 3. Now let $z \in Z$. Theo. 3 yields $d(o, z) = r$. If $o \preceq z$, then $z \in B_r^{max}(o)$ by Prop. 2. If $o \not\preceq z$, then Prop. 1 yields $d(o, z) > |z| - |o|$, *i.e.*, $|z| < |o| + r$. But then, $d(z, m) \geq |m| - |z| > |m| - |o| - r = d(w, m)$ for all $w \in B_r^{max}(o)$, which contradicts $z \in Z$. So $Z \subseteq B_r^{max}(o)$. □

Not in ECML

Not in ECML

Submitting $(a_1 \dots a_n)^k$ with a sufficiently large k is sure to be corrected by a string from $B_r^{max}(o)$. So all that remains is to find such an interesting k. The

following lemma states that if one asks the Oracle to correct a string made of a lot of a's, then the correction contains precious informations on the radius and the number of occurrences of a's in the centre:

Lemma 3. *Consider the ball $B_r(o)$ and let $a \in \Sigma$ and an integer $j \in \mathbb{N}$ s.t. $a^j \notin B_r(o)$. Let $w = \text{CQ}(a^j)$. If $|w| < j$, then $|w|_a = |o|_a + r$.*

Proof. Let $a^j \notin B_r(o)$ and $w = \text{CQ}(a^j)$. By Theo. 3, we get $d(o,w) = r$ and $d(w,a^j) = d(o,a^j) - r$. As $|w| < |a^j|$, the computation of $d(w,a^j)$ consists in (1) substituting all the letters of w that are not a's, and (2) doing insertions of a, so $d(w,a^j) = j - |w|_a$. Following the same arguments, $d(o,a^j) = j - |o|_a$. So $j - |w|_a = j - |o|_a - r$, that is to say, $|w|_a = |o|_a + r$. □

Not in ECML

Not in ECML

Now, let us assume that the alphabet is $\Sigma = \{a_1, \ldots, a_n\}$ and let $j_1, \ldots, j_n \in \mathbb{N}$ be large integers. Define $k = \sum_{i=1}^{n} |\text{CQ}(a_i^{j_i})|_{a_i}$. Then, Lemma 3 brings $k = \sum_{i=1}^{n} (|o|_{a_i} + r) = |o| + |\Sigma| \cdot r \geq |o| + r$. Thus we can plug k into Lemma 2 to get a string $w = \text{CQ}\left((a_1 \ldots a_n)^k\right) \in B_r^{max}(o)$. Moreover, due to Prop. 2, we have $|w| = |o| + r$. Therefore, as $k = |o| + |\Sigma| \cdot r$ and $|w| = |o| + r$, we deduce that $r = (k - |w|)/(|\Sigma| - 1)$ (and $|o| = (|\Sigma| \cdot |w| - k)/(|\Sigma| - 1)$). In other words, we can guess the radius (and the length of the centre).

Let us summarise, by assuming that $\Sigma = \{a_1, \ldots, a_n\}$ and that the target is the ball $B_r(o)$. (1) For each letter a_i, the Learner asks for the correction of a_i^j where j is sufficiently large to get a correction whose length is more than j; (2) We define $k = \sum_{i=1}^{n} |\text{CQ}(a_i^{j_i})|_{a_i}$ and suppose the Learner gets the correction w for the string $m = (a_1 \ldots a_n)^k$; (3) From k and $|w|$, we deduce r; (4) The Learner uses EXTRACT_CENTRE on w and r in order to find o. In other words, we are able to learn the balls with CQ:

Algorithm 2 Identification of the balls using CQ

Require: The alphabet $\Sigma = \{a_1, \ldots, a_n\}$
Ensure: The representation (o, r) of the ball $B_r(o)$
1: $j \leftarrow 1; k \leftarrow 0$
2: **for** $i = 1$ **to** n **do**
3: **while** $|\text{CQ}(a_i^j)| \geq j$ **do** $j \leftarrow 2 \cdot j$ **end while**
4: $k \leftarrow k + |\text{CQ}(a_i^j)|_{a_i}$
5: **end for**
6: $w \leftarrow \text{CQ}((a_1 a_2 \ldots a_n)^k)$
7: $r \leftarrow (k - |w|)/(|\Sigma| - 1)$
8: $o \leftarrow$ EXTRACT_CENTRE(w, r)
9: **return** (o, r)

For instance, consider the ball $B_2(bb)$ defined over $\Sigma = \{a, b\}$. Our algorithm begins by looking for the corrections of a^j and b^j with a sufficiently large j. We might observe: $\text{CQ}(a) = \text{YES}$, $\text{CQ}(a^2) = \text{YES}$, $\text{CQ}(a^4) = aabb$, $\text{CQ}(a^8) = abba$, $\text{CQ}(b^8) = bbbb$. So $k = |abba|_a + |bbbb|_b = 2 + 4 = 6$. Then $\text{CQ}\left((ab)^6\right) =$

CQ($ababababab$) = $baba$, for instance, so $r = (6-4)/(2-1) = 2$. Finally, EXTRACT_CENTRE($baba, 2$) returns bb. So Algo. 2 returns $(bb, 2)$.

Theorem 5. *Given any fixed polynomial $q()$, the set $\mathcal{GB}_{\mathrm{q}}(\Sigma)$ of all q-good balls $B_r(o)$ is identifiable with an algorithm using $\mathcal{O}(|\Sigma| + |o| + r)$ CQ and a polynomial amount of time.*

Proof. The identifiability is clear. Concerning the complexity, the corrections of the strings a_i^j requires $\mathcal{O}(|\Sigma| + \log(|o| + r))$ CQ (lines 2-5). EXTRACT_CENTRE needs $\mathcal{O}(|o| + r)$ CQ (line 8). □

Notice that the set of all the balls, that contains good balls but also *huge* ones *s.t.* $r > 2^{|o|}$ for instance, is not polynomially identifiable with our algorithm since $\mathcal{O}(|\Sigma| + |o| + r) > \mathcal{O}(2^{|o|})$ for some of them.

5 Experiments with a Human-like Oracle

In this section, we would like to show the advantages of our approach. Therefore, we made several experiments that aim at studying the responses of our algorithm faced with an Oracle that could be human. As we said in introduction, our algorithm (1) should make use of as few queries as possible (since the Oracle gets quickly 'tired') and (2) should not believe unwisely the answers he gets (since they can be approximate).

5.1 On the Practical Number of CQ

Not in ECML

We have seen that the balls are identifiable with a linear number of CQ. This may be too much in a human setting. Nevertheless, this complexity corresponds to the worst case. Moreover, we are going to show that this bound is rarely reached in practice.

By Theo. 5, the number of CQ needed to identify $B_r(o)$ is $\mathcal{O}(|\Sigma| + |o| + r)$. So, if we fix $|o| + r = 1000$, the required number of CQ should be globally constant. Therefore, we fix $\Sigma = \{a, b\}$, and make r vary between 0 and 1000. For every value of r, we randomly choose 100 centres o *s.t.* $|o| = 1000 - r$. Then we feed the Oracle with the 100 balls $B_r(o)$, and we count the number of CQ used by our algorithm to identify them. In Fig. 1, we show the average number of CQ that we need for each radius.

Basically, the number of CQ is not constant but grows quasi-linearly between 500 and 1000 with the radius. The extreme points of this curve are easy to justify. Recall that our algorithm uses the function EXTRACT_CENTRE. Besides, a careful reading of the proof of Theo. 5 shows that the cost of this function is the essential part of the complexity of our algorithm: Without it, we could expect a logarithmic complexity, that is, $\mathcal{O}(|\Sigma| + \log(|o| + r))$.

EXTRACT_CENTRE finds o by scanning a string $w \in B_r^{max}(o)$. By Prop. 2, $|w| = |o| + r = 1000$ and $o \preceq w$. Therefore, when $r \simeq 1$ and $|o| \simeq 999$, EXTRACT_CENTRE must scan on average half of the strings w to find the only

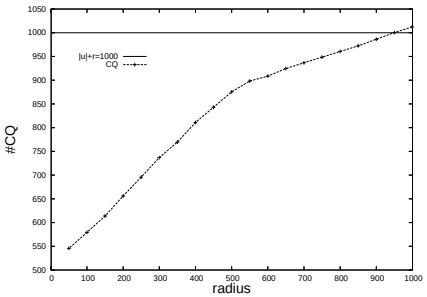


Fig. 1. Number of CQ needed to identify a ball $B_r(o)$ when $|o| + r = 1000$. For each r, we compute the average over 100 balls.

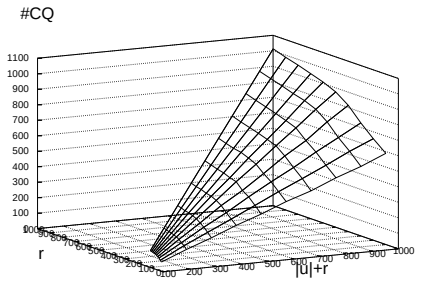


Fig. 2. Number of CQ needed to identify a ball $B_r(o)$ when $|o| + r = t$ with $t \in [100; 1000]$. For each r, we compute the average over 100 balls.

inserted letter in o. So he is going to submit $\simeq 500$ CQ. On the other hand, when $r \simeq 999$ and $|o| \simeq 1$, as EXTRACT_CENTRE must swap r letters to find o, he uses $\simeq 1000$ CQ. Notice that in the latter case, the curve of Fig. 1 shows a number of CQ that is a bit more than 1000, about 1012. This difference is due to the first stages of the algorithm, in which is used the logarithmic number of CQ.

Now, the linearity of the curve, as well as the change in the slope when $r \simeq |o| \simeq 500$, are much more difficult to justify. If we had worked with a larger alphabet Σ, then EXTRACT_CENTRE would have made $\simeq 1000 \cdot r/(r+1)$ CQ to get o. But in our setting, with a small alphabet, there are plenty ways to find o as a subsequence of w. In consequence, our rough formula becomes wrong, and formally assessing the number of CQ needed to extract o, that would justify the curve, is an intricate combinatorial problem. We can only note that this number is often much smaller than $1000 \cdot r/(r+1) \simeq 1000$ when $100 \leq r \leq 900$.

At last but not least, in order to check that the behaviour above was not due to the arbitrary hypothesis $|o|+r = 1000$, we have repeated the same experiment with $|o| + r = t$, where t takes values ranging from 100 and 1000 (see Fig. 2). The surface we get confirms our conclusions: The average number of CQ is less than the theoretical one for big centres and small radii. Thus, our algorithm is efficient under these conditions.

Not in ECML

5.2 Faced with an Approximate Oracle

Our algorithm was designed in an ideal setting, where we have assumed that the Oracle was a perfect machine: Her answers were so precise that we could scrupulously characterise them (see Theo. 3). However, in practice, an Oracle is often an expert, a human being, and in such settings, our assumption is no longer correct. Indeed, computing the correction of a^{500} *w.r.t.* the ball $B_{500}(ab^{500})$ is

probably out of the cognitive capacities of any human being. In this section, we would like to show, with a series of experiments, that our algorithm withstands such approximate (*i.e.*, inaccurate, noisy) answers.

Modelling the Approximate Oracle

We want here to design an Approximate Oracle that might look like a human being. So let us consider a string w and a ball $B_r(o)$. Let $\text{CQ}_h(w)$ denote the answer of the Approximate Oracle, and $\text{CQ}(w)$ the answer that would give a Perfect Oracle (as before).

First, we assume that an expert can easily determine whether an example fulfils a concept or not, thus here, whether w belongs to $B_r(o)$ or not. So we assume that if $\text{CQ}(w) = \text{YES}$, then $\text{CQ}_h(w) = \text{YES}$. Secondly, what is really hard for the expert is to *compute* the correction of w when $w \notin B_r(o)$, and more precisely, a string of the ball that is *as close to* w *as possible*. Again, it is reasonable to think that $\text{CQ}_h(w)$ will be in the ball, so $d(o, \text{CQ}_h(w)) \leq r$. But often, $\text{CQ}_h(w)$ will not be as close as possible to w, so $d(w, \text{CQ}_h(w)) \geq d(w, \text{CQ}(w))$. Staying a step ahead, we must now quantify how far $\text{CQ}_h(w)$ could be from w, *w.r.t.* a perfect correction.

A very first idea is to use a geometric distribution. So let $X = d(w, \text{CQ}_h(w)) - d(w, \text{CQ}(w))$, that measures how far an approximate correction is from a perfect one. Intuitively, an approximate but strong Oracle will often provide corrections *s.t.* $X = 0$, sometimes $X = 1$ and rarely $X \geq 2 \ldots$ Conversely, an Adversary, that does not want us to learn, would systematically give bad answers *s.t.* $X > 0$. To formalise this idea, we introduce a confidence parameter $0 < p \leq 1$, called the *accuracy level of the Oracle*, that translates the quality of her answers. The accuracy of a Perfect Oracle will be 1, whereas the accuracy of a Malicious Adversary would tend to 0.

Concerning the distribution for X, we assume that $Pr(X = k) = (1-p)^k p$, for all $k \in \mathbb{N}$. Therefore, with probability $(1-p)^k p$, the correction $\text{CQ}_h(w)$ of a string w will be in the target ball, at distance k of $\text{CQ}(w)$. Basically, we get $E(X) = \sum_{k=0}^{\infty} kp(1-p)^k = (1-p)\sum_{k=1}^{\infty} kp(1-p)^{k-1} = (1-p)(1/p) = (1/p) - 1$. So when the Oracle is very accurate, say $p = 0.8$, then the average distance between an approximate and a perfect correction is low (0.25). Conversely, an expert with limited computation capacities, say $p = 0.4$, might often provide inaccurate corrections, at distance 1.5 on average.

Our model of Approximate Oracle is simple. For instance, we do not suppose that she has any memory, thus by submitting twice every string w, we will probably get 2 different corrections, that could be used to *correct the corrections*! In other words, as we assume the independence of her answers, multiplying the same query allows to *boost* our Approximate (Weak) Oracle into a Perfect (Strong) one. On the other hand, we want here to study the behaviour and the resistance of our algorithm to approximate answers, not to design the best possible algorithm, able to assimilate inaccurate corrections. So our simple model of Approximate Oracle is sufficient from this viewpoint.

Behaviour of the Algorithm faced with an Approximate Oracle

Following Theo. 5, our algorithm systematically guesses the target ball with the help of a Perfect Oracle. Thus ideally, he does not commit any error. But of course, this is wrong when he is in front of an Approximate Oracle. There are several ways to measure his error. We think that the statistical methods must be discarded since they aim at assessing the generalisation error using test sets and cross-validation for instance, that is far from our learning paradigm: The spirit of Query Learning is the exact identification of all the balls.We want here to show that Algo. 2 is resistant to approximate corrections. Therefore, a more relevant measure is the *precision* of the algorithm, thus the number of balls he is able to find over the number of balls he is asked to retrieve.

We have conducted the following experiment: We fix $|o| + r = 200$; For every $0.5 \leq p \leq 1$, we make r vary between 20 and 180 by steps of 20; Then we assess the precision of the algorithm over 10 balls whose centres, of length $200 - r$, are randomly chosen. We get Fig. 3. We remark that the algorithm is pretty robust to the approximations of the Oracle. For instance, he is able to identify about 75% of the balls faced with an accuracy level of $p = 0.9$. Of course, as one can expect, with lower levels of accuracy, the performance of the algorithm quickly drops (15% for $p = 0.5$).

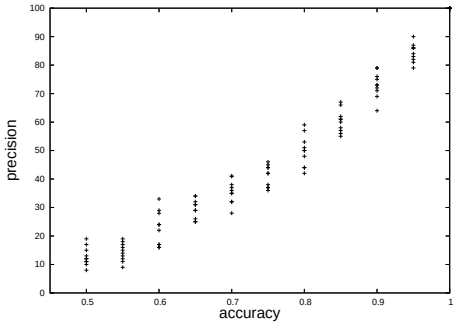


Fig. 3. Precision of the algorithm faced with an Approximate Oracle in function of the accuracy level p. For each $r \in [20; 180]$, we compute the average over 10 balls whose length of the centres is $200 - r$.

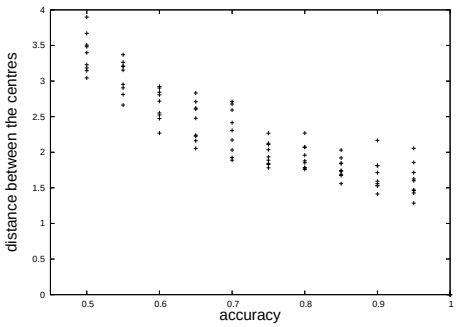


Fig. 4. Average distances between the centres of the target balls and the centres of the learnt balls, when the algorithm fails in retrieving them.

We also show, in Fig. 4, the average distances between the centres of the target balls and the centres of the balls learnt by the algorithm when he fails to find them. We observe that these distances are not that important: Even with an accuracy level of $p = 0.5$, this distance is less than 4. Therefore, there is reason to hope that with only local modifications of the returned centres, and without doing more CQ, we may be able to improve the learning results.

Improving the Precision with a posteriori *Heuristics*

We have seen that our algorithm was able to assimilate the approximations of the Oracle up to a certain level of accuracy. Moreover, the learnt centre returned by the algorithm is generally not far from the target one. Thus, it is reasonable to think that we could find the target centre by mining the neighbourhood of the learnt one, using local edit modifications. These kind of approaches has been pioneered by Kohonen in [19] and is surveyed in [20].

More precisely, suppose that the learnt ball is $B_r(o)$ and let $\mathcal{Q}$ denote the set of all the corrections the Approximate Oracle has returned during the run of the Learner. Consider the set $B_1(o)$ of all the strings that one gets by applying one edit operation on o. Some of them may be the centre of balls that contain all the corrections of $\mathcal{Q}$ but are smaller than $B_r(o)$, in terms of radius; Following the usual Occam's razor arguments, they appear to be more relevant than $B_r(o)$. So, as a first criterion, we should focus on these centres. We define $r' = \min_{o' \in B_1(o)} \left(\max_{w \in \mathcal{Q}} d(o', w) \right)$ and H'_1 the subset of $B_1(o)$ that induce r' as minimum radius.

Experimentally, we have observed that H'_1 could contain a lot of strings, so we need a second criterion to choose one of them. Suppose that the Approximate Oracle is not too bad. If $B_{r'}(o')$ is the target ball, then most of the corrections she will provide, will be on the circle delimiting this ball, by Theo. 3. So choosing the centres $o' \in H'_1$ which maximise the number of strings in $\mathcal{Q}$ at distance r' from o' seems a good idea. Let H_1 denote this set.

Again, even though the set of possible centres dramatically reduces *w.r.t.* $B_1(o)$, it is still possible that H_1 contains several strings. Therefore, if H_1 is a singleton, we return it; Else if o is in H_1, we return o; Else we randomly return one string of H_1.

These heuristics will be very good each time o is at distance 1 from the target centre. But as soon as this distance grows, the algorithm will fail again. In order to enhance these one-step heuristics, we can consider that the construction of H_1 is the first stage of an iterative process and design a second until-convergence heuristic as follows: Assume that H_i was built after i iterations; We define H_{i+1} by (1) computing all the strings at distance 1 from the strings of H_i and (2) among them, keeping those that verify the same 2 criteria as those used for H_1; We repeat the process until $H_{i+1} = H_i$ and ultimately choose the final centre as for H_1.

In order to show that the balls learnt by our algorithm can be corrected *a posteriori*, we compare, in a series of experiments, the precision of the algorithm without any post-treatment, with the one-step heuristics and with the until-convergence heuristics. We fix $|o| + r = 200$. We make the accuracy level vary from 0.5 to 1 and the radius, from 20 to 180. For each pair (accuracy, radius), we randomly draw 50 centres of length $200 - r$, and test our algorithm. We plot the resulting average precisions in Fig. 5.

We can remark that whatever the accuracy level, using the until-convergence heuristics is never worse than the one-step heuristics, that is never worse than

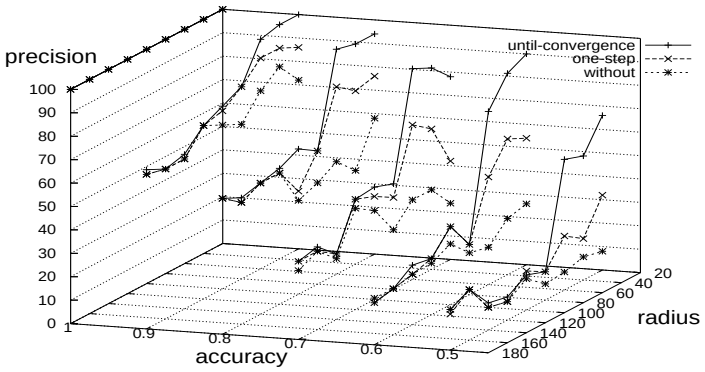


Fig. 5. Precision of the algorithm with and without heuristics in function of accuracy and radius when $|o|+r = 200$. For each pair (accuracy, radius), we compute the average over 50 balls.

no post-treatment at all. Nevertheless, our heuristics do not always improve the precision of the algorithm: This depends on the radius of the target ball (thus also on the length of the centre). In order to detail this, we have extracted 2 transverse sections, shown in Fig. 6, where we fix the radii.

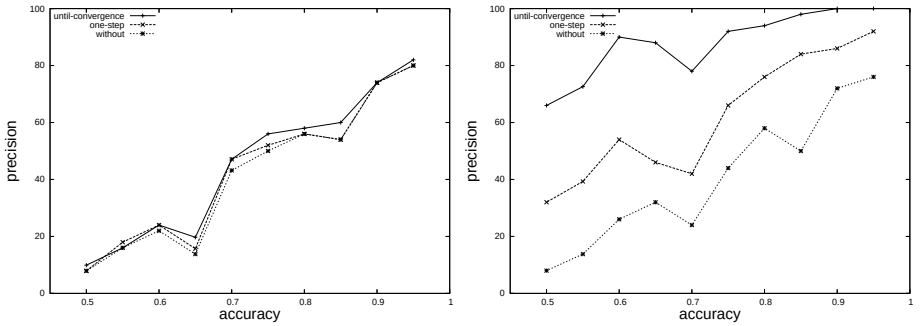


Fig. 6. Precision of the algorithm when $|o| + r = 200$ for $r = 120$ (left) and $r = 20$ (right). For each accuracy, we compute the average over 50 balls.

The left curves of Fig. 6 describe the precision of the algorithm for target balls such that $r = 120$ and $|o| = 80$. In this case, we earn hardly anything using the heuristics. This is probably due to the fact that the size of the set $\mathcal{Q}$, which is used to control the heuristics, is incomparably smaller than the volume of such balls. In other words, the heuristics are not sufficiently guided by $\mathcal{Q}$ towards the targets, because $\mathcal{Q}$ is not informative enough.

On the other hand, the right curves of Fig. 6 describe the precision for target balls such that $r = 20$ and $|o| = 180$. Basically, our heuristics outperform the precision *w.r.t.* the algorithm without any post-treatment, whatever the accuracy level of the Oracle. Moreover, the benefit is all the more important since the accuracy level is bad. For instance, when $p = 0.6$, the until-convergence heuristics is able to dramatically boost the precision from 26% to 90%.

So in this setting, faced with an Approximate Oracle, and with no further enhancement, our algorithm produces balls that are so close to the targets that they can easily be improved using only basic local modifications.

6 Discussion and Conclusion

In this work, we have used correction queries to learn languages from an Oracle. The intended setting is that of an inexact Oracle, and experiments show that the algorithm we propose can learn a language sufficiently close to the target for simple local modifications (with no extra queries) to be possible. In order to do this, the languages we consider are balls of strings defined with the edit distance. Beyond their outward simplicity, they often have intricate and counter-intuitive properties. Studying them allowed us to catch a glimpse of the geometry of Σ^*, which is very different from the Euclidean geometry. A number of questions and research directions are left open by this work:

A first question concerns the distance we use. We have chosen to work with the unitary edit distance, but in many applications, the edit operations can have different weights, depending on how easy they are or how often they occur. Preliminary work has allowed us to notice that the algorithmics could change considerably depending on the sorts of weights we used; For instance with the substitutions costing less than the other 2 operations a much faster algorithm exists, requiring only $\mathcal{O}(\log(|o| + r))$ queries [21].

A second question concerns the inaccuracy model we are using: As noticed in Sect. 5, with the current model it would be possible to repeat the same CQ various times, getting different corrections, but possibly being able, through some majority vote scheme, to get the adequate correction with very little extra cost. Just asking for *persistent* corrections is not enough to solve this problem: A good model should require that if one queries from a close enough string (a^{999} instead of a^{1000}) then the corrections should also remain close. Topologically, we would expect the Oracle to be k-Lipschitz continuous (with $0 < k < 1$).

A third more challenging problem then arises: Our choice here was to learn supposing the Oracle was exact, and correcting later. But a more direct approach might be better, by taking into account the inexactitude of the Oracle when interpreting the correction.

These are some of the directions that should be followed in order to hope to learn from correction queries to an imperfect oracle.

Acknowledgement The authors wish to thank Jose Oncina and Rémi Eyraud for their help in the preliminary phase of this work and Dana Angluin for helpful comments.

References

1. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR **163**(4) (1965) 845–848
2. Wagner, R., Fisher, M.: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–178
3. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press (1998)
4. Amengual, J.C., Sanchis, A., Vidal, E., Benedí, J.M.: Language simplification through error-correcting and grammatical inference techniques. Machine Learning Journal **44**(1) (2001) 143–159
5. Miclet, L.: Grammatical inference. In: Syntactic and Structural Pattern Recognition - Theory and Applications. Word Scientific (1986) 237–290
6. Bernard, M., Janodet, J.C., Sebban, M.: A discriminative model of stochastic edit distance in the form of a conditional transducer. In: Proc. of the 8th Int. Coll. in Grammatical Inference (ICGI'06), LNCS 4201 (2006) 240–252
7. Oncina, J., Sebban, M.: Learning stochastic edit distance: Application in handwritten character recognition. Pattern Recognition **39**(9) (2006) 1575–1587
8. Tantini, F., de la Higuera, C., Janodet, J.C.: Identification in the limit of systematic-noisy languages. In: Proc. of the 8th Int. Coll. in Grammatical Inference (ICGI'06), LNCS 4201 (2006) 19–31
9. Mukouchi, Y., Sato, M.: Refutable language learning with a neighbor system. In: Proc. of the 12th Int. Conf. on Algorithmic Learning Theory (ALT'01), LNAI 2225 (2001)
10. Kearns, M.J., Li, M.: Learning in the presence of malicious errors. SIAM Journal of Computing **22**(4) (1993) 807–837
11. Crochemore, M., Hancart, C., Lecroq, T.: Algorithmique du texte. Vuibert (2001)
12. Schulz, K.U., Mihov, S.: Fast string correction with levenshtein automata. Int. Journal on Document Analysis and Recognition **5**(1) (2002) 67–85
13. Angluin, D.: Queries and concept learning. Machine Learning Journal **2** (1987) 319–342
14. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75** (1987) 87–106
15. de la Higuera, C.: Data complexity issues in grammatical inference. In: Data Complexity in Pattern Recognition. Springer-Verlag (2006) 153–172
16. Angluin, D.: Queries and concept learning. Machine Learning **2** (1988) 319–342
17. Becerra-Bonache, L., Yokomori, T.: Learning mild context-sensitiveness: Towards understanding children's language learning. In: Proc. of the 7th Int. Coll. in Grammatical Inference (ICGI'04), LNCS 3264 (2004) 53–64
18. de la Higuera, C., Casacuberta, F.: Topology of strings: median string is NP-complete. Theoretical Computer Science **230** (2000) 39–48
19. Kohonen, T.: Median strings. Information Processing Letters **3** (1985) 309–313
20. Martínez-Hinarejos, C.D., Juan, A., Casacuberta, F.: Use of median string for classification . In: ICPR. (2000) 2903–2906
21. Bonache, L.B., de la Higuera, C., Janodet, J.C., Tantini, F.: Apprentissage des boules de mots avec des requêtes de correction (*in french*). In: Proc. of 9th Conférence en Apprentissage, Presses Universitaires de Grenoble (2007)